7 Class Super Operator (super_op)

7.1 Overview

The *Class SUPER OPERATOR* defines all the necessary attributes of a quantum mechanical superoperator, **LOp**. Within *Class SUPER OPERATOR* are also specifications of superoperator properties (dimension, ...), algebras (+, *,...), and definitions of all available superoperator functions.

7.2 Available Superoperator Functions

Superoperator Basic Functions

super_op	- Constructor:	LOp, LOp(mx), LOp(mx1, mx2), LOp(mx, bs), LOp(LOp1).	page 125
+	- Addition:	LOp + LOp, LOp + mx, mx + LOp.	page 127
+=	- Unary Addition:	+= LOp, += mx.	page 127
-	- Subtraction:	LOp - LOp, LOp - mx, mx - LOp.	page 128
-=	- Unary Subtraction:	-= LOp, -= mx.	page 129
*	- Multiplication:	LOp * LOp, LOp*mx, mx*LOp, LOp*Op,z*LOp,LOp*z.	page 130
*=	- Unary Multiplication:	*=LOp, *=mx, *=z.	page 131
-	- Negation:	-LOp.	page 128
=	- Assignment:	= LOp, = mx.	page 126
/	- Division:	LOp / z.	page 131
/=	- Unary Division:	/= z.	

Complex Superoperator Functions[1]

left	- Left Translation LOp:	Op, mx	page 133
right	- Right translation LOp:	Op, mx	page 134
commutator	- Commutation Lop:	[Op,], [mx,]	page 135
d_commutator	- Double commutation LOp:	[Op, [Op,]], [mx, [mx,]], [Op1, [Op2,]], [mx1, [mx2,]]	page 136
U_transform	- Unitary transform LOp:	U__U^{-1} : LOp*Op = U*Op*U^{-1}	page 137
exp	- Superoperator exponential:	exp(LOp);	page 138

Superoperator Internal Access

get_mx	- Retrieve superoperator matrix:	LOp
put_mx	- Input superoperator matrix:	LOp
put_basis	- Input superoperator basis:	LOp
()	- Direct LOp element access:	z = LOp(3,2).

1. The commutation and double commutations superoperators may take spin operators instead of operators or matrices. For the unitary transformation superoperator U is either an operator, spin operator, or matrix.

put	- Input LOp element directly:		
get	- Direct LOp element access:		

Basis Manipulations

set_EBR	- Put LOp into its eigenbasis:	LOp	page 139
set_HBR	- Put LOp into its default Liouville space basis.		page 139
Op_base	- Put LOp into the Liouville space basis of another superoperator.		
	- Put operator into the Hilbert space basis of a superoperator.		

Superoperator I/O

<<	- Send LOp to an output stream:	LOp	page 139

7.3 Arithmetic Operators

7.3.1 super_op

Usage:

```
#include <super_op.h>
super_op ( )
super_op (matrix &mx)
super_op (matrix &mx1, matrix &mx2)
super_op (matrix &mx, basis &bs)
super_op (super_op &LOp)
```

Description:

The function *super_op* is used to create a superoperator quantity.

1. super_op() - sets up an empty superoperator which can later be explicitly specified.
2. super_op(mx) - sets up a superoperator with the matrix in the argument assumed to be the superoperator in the default basis. The specified matrix must be a square array in the Liouville space.
3. super_op(mx1, mx2) - With two matrices as arguments, the function sets up a superoperator with matrix representation **mx**1 in the basis formed from matrix **mx2**. Again, **mx**1 must be a square matrix of the Liouville space dimension. The basis should relate properly to the default basis, and be the Hilbert space dimension.
4. super_op(mx, bs) - With a matrix (**mx**) and a basis (**bs**) as arguments, the function sets up a superoperator with matrix representation **mx** in the basis **bs**. Again, **mx** must be a square matrix and its dimension that of the Liouville space. The basis should relate properly to the default basis, and be the Hilbert space dimension.
5. super_op(LOp) - One may produce a superoperator from another superoperator. The new superoperator will then be equivalent to the input superoperator **LOp**.

Return Value:

Creates a new superoperator which may be subsequently used with all defined superoperator functions.

Examples:

```
#include <super_op.h>
matrix mx;
basis bs;
super_op LOp;                 // produces an empty superoperator LOp.
super_op LOp1(mx);            // set superoperator LOp, matrix mx in default basis.
super_op LOp2(mx, bs);        // set superoperator LOp2, matrix mx in the basis bs.
super_op LOp3(LOp2);          // set superoperator LOp3 equal to current LOp2.
```

Mathematical Basis:

Each superoperator is a matrix in Liouville space and has associated with it a basis in Hilbert space.

Superoperator Matrix in Liouville Space

11	12	13	. . . . .	$1N^2$
21	22	23	. . . . .	$2N^2$
31	32	33	. . . . .	$3N^2$
41	42	43	. . . . .	$4N^2$
.	.	.		.
.	.	.		.
.	.	.		.
.	.	.		.
.	.	.		.
.	.	.		.
N^21	N^22	N^23	. . . . .	N^2N^2

Superoperator Basis in Hilbert Space

11	12	13	. . .	1N
21	22	23	. . .	2N
31	32	33	. . .	3N
.	.	.		.
.	.	.		.
.	.	.		.
.	.	.		.
N1	N2	N3	. . .	NN

In all superoperator constructors the matrix provided must be in the Liouville space dimension, the Hilbert space dimension squared. The constructor basis (or matrix to be used as the basis) must be of Hilbert space dimension, smaller in size than the actual superoperator matrix.

See Also: =

7.3.2 =

Usage:

```
#include <super_op.h>
super_op operator = (super_op& LOp)
super_op operator = (matrix& mx)
```

Description:

This allows for the ability to equate either a superoperator or a matrix to another superoperator.

1. For the equating of two superoperators, **LOp** = **LOp1**, superoperator **LOp** is set equal to superoperator **LOp1**. **LOp** will be in the current basis of **LOp1**.
2. For the assignment of a matrix to a superoperator, **LOp** = **mx**, the superoperator **LOp** is set equal to the matrix **mx** and the basis is assumed to be the default basis. The matrix **mx** must be in the Liouville space.

Return Value:

none.

Example(s):

```
#include <super_op.h>
matrix mx;                    // define a matrix mx.
```

```
super_op LOp1, LOp2;          // define two superoperators LOp1 and LOp2.
LOp1 = LOp2;                  // LOp1 set equal to superoperator LOp2.
LOp1 = mx;                    // LOp1 set in the default basis to have matrix mx.
```

See Also: super_op (constructors)

7.3.3 +

Usage:

```
#include <super_op.h>
super_op operator + (super_op& LOp1, super_op& LOp2)
super_op operator + (super_op& LOp, matrix& mx)
super_op operator + (matrix& mx, super_op& LOp)
```

Description:

This allows for the addition of two superoperators, **LOp1** + **LOp2** and the addition of a superoperator with a matrix, **LOp1** + **mx**.

1. LOp1 + LOp2 - Definition of the addition of two superoperators **LOp1** and **LOp2**. A check is made to insure that both superoperators are in the same basis. If this is not true, superoperator **LOp2** is transformed into the basis of superoperator **LOp1** prior to the addition, thus insuring that the addition *produces a result in (and only in) the same basis of* **LOp1**.
2. LOp + mx - Definition of the addition of a matrix mx to a superoperator **LOp**. The matrix mx is assumed to be a matrix in the default basis and the addition takes place in the default basis. Superoperator **LOp** is first placed in the default basis, the addition takes place, and then the result is *new superoperator in the default basis.*
3. mx + LOp - Definition of the addition of a superoperator **LOp** to a matrix mx. The result is equivalent to the previous addition, it produces a *new superoperator in the default basis*.

Return Value:

A new superoperator which exists in an appropriate representation.

Example(s):

```
#include <super_op.h>
matrix mx;
super_op LOp1,LOp2,LOp3;      // define three superoperators LOp1,LOp2, & LOp3.
LOp3 = LOp1 + LOp2;           // LOp3 is sum LOp1 + LOp2. Only in WB of LOp1.
LOp3 = mx + LOp1;             // LOp3 is sum LOp1 + matrix mx. Only in DB.
LOp3 = LOp1 + mx;             // same as the previous line.
```

See Also:

+=, -, -=

7.3.4 +=

Usage:

```
#include <super_op.h>
super_op operator += (super_op &LOp)
super_op operator += (matrix &mx)
```

Description:

This allows for the addition of two superoperators of the type **LOp = LOp** + **LOp1** or the addition of a matrix to a superoperator **LOp = LOp** + **mx**.

1. LOp += LOp1 - Definition of the unary addition of two superoperators **LOp** and **LOp1**. A check is made to insure that both superoperators are in the same basis. If this is not true, **LOp1** is placed into the basis of **LOp** prior to the addition, thus insuring that the subtraction *produces a result in (and only in) the same basis of* **LOp**.
2. LOp += mx - Unary addition of a matrix **mx** to a superoperator **LOp**. The matrix **mx** is assumed to be a superoperator matrix in the default basis and the addition takes place in the default basis. Superoperator **LOp** is first placed in the default basis, the matrix **mx** is then added to it.

Use of this operator is more computationally efficient that the two step operation.That is, the statement **LOp** += **LOp1**; is preferred over the statement **LOp** = **LOp** + **LOp1**;.

Return Value:

A new superoperator which exists in an appropriate representation.

Example(s):

```
#include <super_op.h>
matrix mx;
super_op LOp, LOp1;          // define two superoperators LOp and LOp1.
LOp += LOp1;                 // LOp1 added to LOp. Only in the WB of LOp.
```

See Also: +, -, -=

7.3.5 -

Usage:

```
#include <super_op.h>
super_op LOp - (super_op& LOp, super_op& LOp1)
super_op LOp - (super_op& LOp, matrix& mx)
super_op LOp - (matrix& mx, super_op& LOp)
super_op LOp - (super_op& LOp)
```

Description:

This allows for the subtraction of two superoperators **LOp** and **LOp1**, for the subtraction of a matrix from a superoperator, and for the negation of a superoperator.

1. LOp - LOp1 - Definition of the subtraction of two superoperators **LOp** and **LOp1**. A check is made to insure that both superoperators are in the same basis. If this is not true, **LOp1** is placed into the basis of **LOp** prior to the subtraction, thus insuring that the subtraction *produces a result in (and only in) the same*

basis of **LOp**.

2. LOp - mx - Definition of the subtraction of a matrix **mx** from a superoperator **LOp**. The matrix **mx** is assumed to be a matrix in the default basis and the subtraction takes place in the default basis. Superoperator **LOp** is first placed in the default basis, the matrix **mx** is then subtracted from it, and then the result is *new superoperator in the default basis*.
3. mx - LOp - Definition of the subtraction of a superoperator **LOp** from a matrix **mx**. The result is equivalent to the negative of the previous subtraction, it produces a *new superoperator in the default basis*.
4. - LOp - Definition of the negation of superoperator **LOp**. The result is a superoperator, in (and only in) the working basis of **LOp,** which is -1.0 * **LOp**.

Return Value:

A new superoperator which exists in an appropriate representation.

Example(s):

```
#include <super_op.h>
matrix mx;
super_op LOp, LOp1, LOp2;      // define three superoperators LOp, LOp1, and LOp2.
LOp2 = LOp - LOp1;             // LOp2 set to LOp - LOp1. Only in the WB of LOp.
LOp2 = LOp -mx;                // LOp2 set to be LOp minus matrix mx. Only in DB
LOp2 = mx - LOp;               // same as -(LOp - mx). Only in the DB of LOp.
LOp2 = -LOp;                   // LOp2 set to negative LOp. Only in the WB of LOp.
```

See Also:+, +=, -=s

7.3.6 -=

Usage:

```
#include <super_op.h>
void operator -= (super_op& LOp1)
```

Description:

This allows for the subtraction of two superoperators of the type **LOp = LOp - LOp1**. A check is made to insure that both superoperators are in the same basis. If this is not true, **LOp1** is transformed into the basis of **LOp** prior to the addition, thus insuring that the subtraction *produces a result in (and only in) the same basis of* **LOp**. Use of this operation is more computationally efficient that the two step operation.That is, the statement **LOp** -= **LOp1**; is preferred over the statement **LOp** = **LOp** - **LOp1**;.

Return Value:

Void. The input superoperator is modified.

Example(s):

```
#include <super_op.h>
matrix mx;
super_op LOp,LOp1;             // define two superoperators LOp and LOp1.
```

```
LOp -= LOp1;                    // LOP set to LOp - LOp1. Only in WB of LOp.
```

See Also:

-,+,+=

7.3.7 *

Usage:

```
#include <super_op.h>
super_op operator * (super_op& LOp1, super_op& LOp2)
super_op operator * (super_op& LOp, matrix& mx)
super_op operator * (matrix& mx, super_op& LOp)
super_op operator * (complex& z, super_op& LOp)
super_op operator * (super_op& LOp, complex& z)
gen_op operator * (super_op& LOp, gen_op &Op)
```

Description:

This allows for the multiplication of two superoperators **LOp1** and **LOp2**, the multiplication of a superoperator and a matrix, and for the multiplication of a scalar and a superperator. Additionally the multiplication of a superoperator into a general operator is defined, but not the reverse.

1. For the multiplication of two superoperators, a check is made to insure that both superoperators are in the same basis. If this is not true, **LOp2** is transformed into the basis of **LOp1** prior to the multiplication, thus insuring that the multiplication *produces a result in the same basis of* **LOp1**. Here, the order of the superoperators can make a difference in the result.
2. For the multiplication of asuperoperator times a matrix, a superoperator is produced. It is assumed that the matrix is a quantity in the default basis so that **LOp** is changed into the default basis before the multiplication.
3. The multiplication of a matrix times a superoperator also produces a superoperator. The treatment is similar to the previous usage except the ordering can make a difference.
4. For the multiplication of a scalar times a superoperator, the complex scalar z is multiplied into each element of **LOp** to produce a superoperator in the same basis of **LOp.**
5. The multiplication of a superoperator times a scalar produces the same result as multiplication of a scalar times a superoperator.
6. The multiplication of a superoperator into a general operator produces a new operator. This operation is performed in the basis of the superoperator.

Return Value:

none.

Example(s):

```
#include <super_op.h>
complex z;
super_op LOp1,LOp2,LOp3;        // define three superperators LOp1, LOp2, and LOp3.
LOp3 = LOp1 * LOp2;             // C is product of LOp1 times LOp2. In WB of LOp1.
```

```
LOp3 = LOp2 * LOp1;          // maybe not same as the previous line, order matters.
LOp3 = LOp1* z;              // LOp3 is LOp1 with all elements multiplied by z.
LOp3 = z * LOp1;             // same result as previous line. Only in WB of LOp1.
gen_op Op1;                  // define two general operators Op1 and Op2.
Op1 = LOp1 * Op2;            // Op1 is LOp1 multiplied into Op2. In WB of LOp1.
```

See Also: *=, +, -, /

7.3.8 *=

Usage:

```
#include <super_op.h>
super_op operator *= (super_op& LOp1)
super_op operator *= (matrix& mx)
super_op operator *= (complex& z)
```

Description:

This allows for the multiplication of two superoperators of the type **LOp1 = LOp1 * LOp2** or of a matrix with asuperoperator, or of a scalar with a superoperator.

1. **LOp2 *= LOp1 -** For the multiplication of two superperators, a check is made to insure that both Operators are in the same basis. If this is not true, **LOp1** is transformed into the basis of **LOp2** prior to the multiplication, thus insuring that the multiplication *produces a result in the same basis of* **LOp2**. Here, the order of the superperators can make a difference in the result.
2. **LOp *= mx -** For the multiplication of a superoperator times a matrix, an superoperator is produced. It is assumed that the matrix is a quantity in the default basis so that **LOp** is changed into the default basis before the multiplication.
3. **LOp *=** z **-** The multiplication of a superoperator times a scalar (complex) also produces a superoperator. This operation *produces a result exclusively in the original working basis of* **LOp**.

Use of this operation is more computationally efficient that the two step operation.That is, the statement **LOp1** *= **LOp2**; is preferred over the statement **LOp1** = **LOp1** * **LOp2**;.

Return Value:

A new superoperator which exists in an appropriate representation.

Example(s):

```
#include <super_op.h>
matrix mx;
super_op LOp1,LOp2;          // define two superoperators LOp1 and LOp2.
LOp1 *= LOp2;                // LOp1 set to LOp1 * LOp2. Only in WB of LOp1.
```

See Also:*, +, +=, -, -=

7.3.9 /

Usage:

```
#include <super_op.h>
super_op operator / (super_op& LOp, complex& z)
```

Description:

This allows for the division of a superoperator by a complex number. Each element of the superoperator matrix is divided by the complex number.The operation *produces a result exclusively in the original working basis of* **LOp**.

Return Value:

A new superoperator which exists in an appropriate representation.

Example(s):

```
#include <super_op.h>
complex z;
super_op LOp;                    // define a superoperator LOp.
LOp *=z;                         // LOp1 set to LOp * z. Only in WB of LOp.
```

See Also: *, +, +=, -, -=

7.4 Complex Functions

7.4.1 left

Usage:

```
#include <super_op.h>
super_op left (gen_op &Op)
```

Description:

Computes the left translation superoperator as defined by equation (14-5),

$$\Gamma \boldsymbol{A} = \boldsymbol{OpA}.$$

The output superoperator is in the working basis of the input operator.

Return Value:

The function returns a superoperator which is the left translation superoperator.

Example(s):

```
#include <super_op.h>
int main ()
 {
  spin_sys sys(1);
 gen_op Op,Op1;                  // construct two null operators
 super_op LOp;                   // construct a null superoperator.
 Op = Fz(sys).                   // set Op to the operator Fz for the spin system.
 Op1 = Fx(sys).                  // set Op1 to the operator Fx for the spin system.
 LOp = left(Op);                 // LOp is the left translation superoperator for Fz.
 cout << LOp*Op1;                // should output matrix Fz*Fx.
 cout << Op*Op1;                 // should also aout Fz*Fx.
 }
```

This example program prints out the following Hilbert space matrix twice, Fz*Fx.

$$\frac{1}{4}\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

Mathematical Basis:

The left translation superoperator is computed from taking cross products of the operator $\boldsymbol{Op}$ with the identiity matrix in accordance with equation (14-6)

$$\Gamma = \boldsymbol{Op} \otimes \boldsymbol{E}$$

7.4.2 right

Usage:

```
#include <super_op.h>
super_op right (gen_op &Op)
```

Description:

Computes the left translation superoperator as defined by equation (14-5),

$$\Gamma \boldsymbol{A} = \boldsymbol{A}\boldsymbol{Op}.$$

The output superoperator is in the working basis of the input operator.

Return Value:

The function returns a superoperator which is the right translation superoperator.

Example(s):

```
#include <super_op.h>
int main ()
 {
  spin_sys sys(1);
 gen_op Op,Op1;                  // construct two null operators
 super_op LOp;                   // construct a null superoperator.
 Op = Fz(sys).                   // set Op to the operator Fz for the spin system.
 Op1 = Fx(sys).                  // set Op1 to the operator Fx for the spin system.
 LOp = right(Op);                // LOp is the right translation superoperator for Fz.
 cout << LOp*Op1;                // should output matrix Fx*Fz.
 cout << Op1*Op;                 // should also aout Fx*Fz.
 }
```

This example program prints out the following Hilbert space matrix twice, Fx*Fz.

$$\frac{1}{4}\begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$$

Mathematical Basis:

The right translation superoperator is computed from taking a cross products of the operator $\boldsymbol{Op}$ with the identiity matrix in accordance with equation (14-6),

$$\Gamma = \boldsymbol{E} \otimes \boldsymbol{Op}^T.$$

7.4.3 commutator

Usage:

```
#include <super_op.h>
super_op commutator (gen_op &Op)
```

Description:

Computes the superoperator equivalent to the commutator of Op as defined by equation (14-3),

$$\Gamma \boldsymbol{A} = [\boldsymbol{Op}, \boldsymbol{A}] = \boldsymbol{OpA} - \boldsymbol{AOp} .$$

The output superoperator is in the working basis of the input operator.

Return Value:

The function returns a superoperator which is the commutation superoperator.

Example(s):

```
#include <super_op.h>
int main ()
 {
  spin_sys sys(1);
 gen_op Op;                  // construct a null operator
 super_op LOp;               // construct a null superoperator.
 Op = Fx(sys).               // set Op to the operator Fx for the spin system.
 LOp = commutator(Op);       // LOp is the commutation superoperator for [Fx, ].
 cout << LOp;                // output commutation superoperator.
 }
```

This example program prints out the commutation superoperator for Fx[1].

$$\begin{bmatrix} 0 & -0.5 & 0.5 & 0 \\ -0.5 & 0 & 0 & 0.5 \\ 0.5 & 0 & 0 & -0.5 \\ 0 & 0.5 & -0.5 & 0 \end{bmatrix}$$

Mathematical Basis:

The commutation superoperator is computed from taking cross products of the operator $\boldsymbol{Op}$ with the identiity matrix in accordance with equation (14-4)

$$\Gamma = (\boldsymbol{Op} \otimes \boldsymbol{E}) - (\boldsymbol{E} \otimes \boldsymbol{Op}^T)$$

1. This matrix is listed in Ernst, Bodenhausen, and Wokaun, page 24, Figure 2.1.2

For a single spin 1/2 particle in the product basis, three commutation superoperators are

$$\hat{I}_x = \frac{1}{2}\begin{bmatrix} 0 & -1 & 1 & 0 \\ -1 & 0 & 0 & 1 \\ 1 & 0 & 0 & -1 \\ 0 & 1 & -1 & 0 \end{bmatrix} \qquad \hat{I}_y = \frac{i}{2}\begin{bmatrix} 0 & -1 & -1 & 0 \\ 1 & 0 & 0 & -1 \\ 1 & 0 & 0 & -1 \\ 0 & 1 & 1 & 0 \end{bmatrix} \qquad \hat{I}_z = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

According to the literature, if the commutator is Hermitiian then the commutator superoperator will be Hermitian as well[1]

7.4.4 d_commutator

Usage:

```
#include <super_op.h>
super_op d_commutator (gen_op &Op)
super_op d_commutator (gen_op &Op1, gen_op &Op2)
```

Description:

Computes the superoperator equivalent to the double commutator of Op. The double commutation superoperator is defined by equation (14-7),

$$\Gamma \boldsymbol{A} = [\boldsymbol{Op}1, [\boldsymbol{Op}2, \boldsymbol{A}]] \ ,$$

where Γ is a superoperator and both $\boldsymbol{Op}1$, $\boldsymbol{Op}2$, and $\boldsymbol{A}$ are general operators

1. d_commutator (gen_op &Op) - The double commutation superoperator if formed in the basis of the operator Op. The operators in the commutator are the same.
2. d_commutator (gen_op &Op1, gen_op &Op2) - T double commutation superoperator is formed from the two input operators. Op2 will be the operator in the inner commutator and Op1 in the outer commutator. The returned superoperator will be in the basis of Op1.

Return Value:

The function returns a superoperator which is the double commutator superoperator.

Example(s):

```
#include <super_op.h>
int main ()
 {
  spin_sys sys(1);
 gen_op Op1, Op2;                  // construct two null operators.
 super_op LOp;                     // construct a null superoperator.
 Op1 = Fp(sys).                    // set Op1 to the operator F+ for the spin system.
 Op2 = Fm(sys).                    // set Op2 to the operator F- for the spin system.
 LOp = d_commutator(Op1);          // double commutation superoperator [Op1[Op1, ].
```

1. Ernst, Bodenhausen, Wokaun, page 20, below equation (2.1.61)

```
LOp = d_commutator(Op1,Op2);    // double commutation superoperator [Op1[Op2, ].
}
```

Mathematical Basis:

The double commutation superoperator is computed from cross products of the operators involved with the identity matrix as given by equation (14-9).

$$\Gamma = \Gamma 1\Gamma 2 = (\boldsymbol{Op}1\boldsymbol{Op}2 \otimes \boldsymbol{E}) - (\boldsymbol{Op}1 \otimes \boldsymbol{Op}2^T) - (\boldsymbol{Op}2 \otimes \boldsymbol{Op}1^T) + (\boldsymbol{E} \otimes \boldsymbol{Op}1^T\boldsymbol{Op}2^T)$$

When both operators are equivalent the equation slightly simplifies to equation (14-10).

$$\Gamma = \Gamma 1\Gamma 1 = (\boldsymbol{Op}^2 \otimes \boldsymbol{E}) - 2(\boldsymbol{Op} \otimes \boldsymbol{Op}^T) + (\boldsymbol{E} \otimes \boldsymbol{Op}^{T^2})$$

Examples of double commutation superoperators for a single spin 1/2 particle in the product basis are given below.

$$\hat{\boldsymbol{I}}_{xx} = [I_x,[I_x,\ ]] = \frac{1}{2}\begin{bmatrix} 1 & 0 & 0 & -1 \\ 0 & 1 & -1 & 0 \\ 0 & -1 & 1 & 0 \\ -1 & 0 & 0 & 1 \end{bmatrix} \qquad \hat{\boldsymbol{I}}_{yy} = [I_y,[I_y,\ ]] = \frac{1}{2}\begin{bmatrix} 1 & 0 & 0 & -1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ -1 & 0 & 0 & 1 \end{bmatrix} \qquad \hat{\boldsymbol{I}}_{zz} = [I_z,[I_z,\ ]] = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\hat{\boldsymbol{I}}_{xy} = [I_x,[I_y,\ ]] = \frac{i}{2}\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & -1 & -1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \qquad \hat{\boldsymbol{I}}_{yz} = [I_y,[I_z,\ ]] = \frac{i}{2}\begin{bmatrix} 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 \end{bmatrix} \qquad \hat{\boldsymbol{I}}_{zx} = [I_z,[I_x,\ ]] = \frac{1}{2}\begin{bmatrix} 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 1 \\ -1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

7.4.5 U_transform

Usage:

```
#include <super_op.h>
super_op U_transform(gen_op &Op)
```

Description:

Computes the superoperator equivalent to the similarity transform (unitary transformation) as defined by equation (14-11),

$$\Gamma\boldsymbol{A} = \boldsymbol{OpAOp}^{-1}.$$

The output superoperator is in the working basis of the input operator.

Return Value:

The function returns a superoperator.

Example(s):

```
#include <super_op.h>
gen_op Op;                  // construct a null operator
super_op LOp;               // construct a null superoperator.
Op = Rx(sys, 90).           // Op to 90 degree x-rotation operator for sys.
```

```
LOp = U_transform(Op);          // LOp unitary transform superoperator Rx __ Rx^-1.
```

Mathematical Basis:

The superoperator in this case relates to the operator $\boldsymbol{Op}$ according to equation (14-12),

$$\Gamma = \boldsymbol{Op} \otimes \boldsymbol{Op}^* ,$$

7.4.6 exp

Usage:

```
#include <super_op.h>
super_op exp(gen_op &LOp)
```

Description:

Computes the exponential of the input superoperator.,

$$\Gamma = exp(\Gamma_1) .$$

The output superoperator is in the basis of the input superoperator.

Return Value:

The function returns a superoperator.

Example(s):

```
#include <super_op.h>
gen_op Op;                      // construct a null operator
super_op LOp;                   // construct a null superoperator.
Op = Fx(sys).                   // Op to Fx for the spin system sys
LOp = d_commutator(Op);         // LOp double commutator {Fx, [Fx, ]].
LOp = exp(Op);                  // LOp exponential of {Fx, [Fx, ]].
```

Mathematical Basis:

7.5 Basis Manipulations

7.5.1 set_EBR

Usage:

```
#include <super_op.h>
super_op set_EBR()
```

Description:

Places the superoperator into its eigenbasis. The superoperator will then be a diagonal array in Liouville space and be associated with a basis transformation array which will convert it back into the original basis in which the superoperator was before the funciton call. The previous representation is destroyed but can be regenerated with the function set_HSBR.

Return Value:

The function is void. It changes the input superoperator into its eigenbasis.

Example(s):

```
#include <super_op.h>
super_op LOp;                  // construct a null superoperator.
LOp.set_EBR();                 // LOp placed into its eigenbasis.
```

Mathematical Basis:

7.5.2 set_HBR

Usage:

```
#include <super_op.h>
super_op set_HBR()
```

Description:

Places the superoperator into its original Hilbert space basis. If for some reason the superoperator basis has been changed from that which it had when formulated, this function will return it to its original basis.

Return Value:

The function is void. It changes the input superoperator into its original Hilbert space basis.

Example(s):

```
#include <super_op.h>
super_op LOp;                  // construct a null superoperator.
LOp.set_HBR();                 // LOp placed into its eigenbasis.
```

Mathematical Basis:

7.5.3 <<

Usage:

```
#include <super_op.h>
ostream& operator << (ostream& str, super_op &LOp)
```

Description:

Sends the superoperator given in the argument list to the output stream specified.

Return Value:

None, the superoperator is put into the output stream

7.6 Description

As the name implies, *Class super_op (SUPER OPERATOR*) deals with superoperators. Throughout this document, use is made of the symbol **LOp** for a superoperator (**Op** for operator and **L** for Liouville space). In mathematical expressions usually the capital gamma is the symbol for denoting a superoperator, Γ.

Basic Structure

Each superoperator is a matrix (*see Class MATRIX*) in Liouville space and has associated with it a basis *(see Class BASIS)* in Hilbert space.

Superoperator Representation Structure

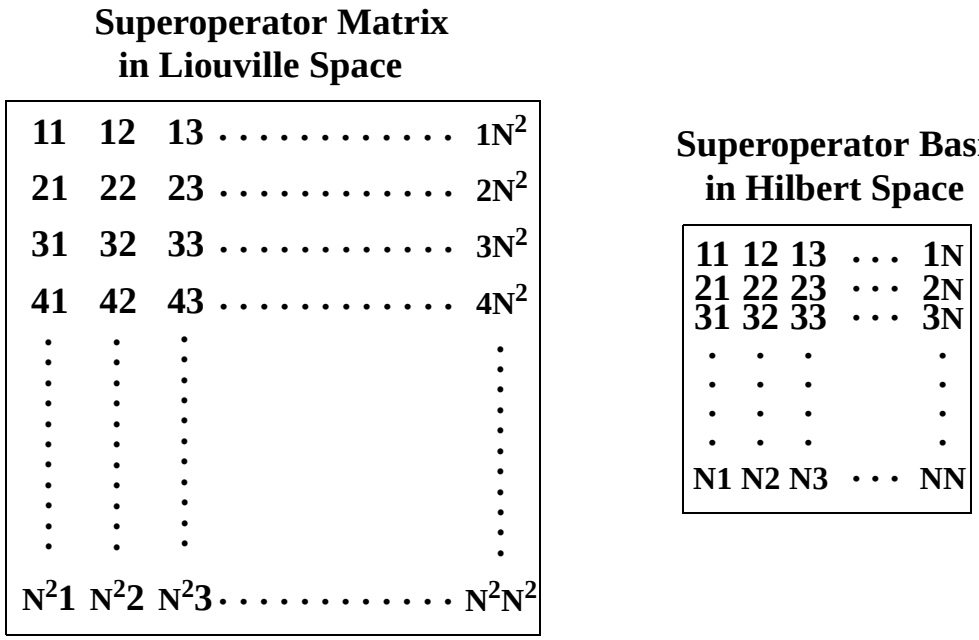

Figure 10-1 - Each superoperator has only one representation: a matrix and a basis. Any change to the representation (basis change) destroys the previous representation.

The matrix form of **LOp** depends upon which basis the superoperator is expressed in. Without knowledge of the basis the superoperator matrix is nearly useless, it could not be freely applied without careful consideration from the external user. *Class super_op* will always associate an **LOp** matrix with an **LOp** basis.

*General Operators as Superoperator*s

There is an intrinsic relationship between general operators (spanning Hilbert space) and superoperators (spanning Liouville space). The following diagram shows the relationship between a general operator, Op, and a superoperator LOp formed (through one of the provided functions) from Op.

Formation of Superoperators from Operators

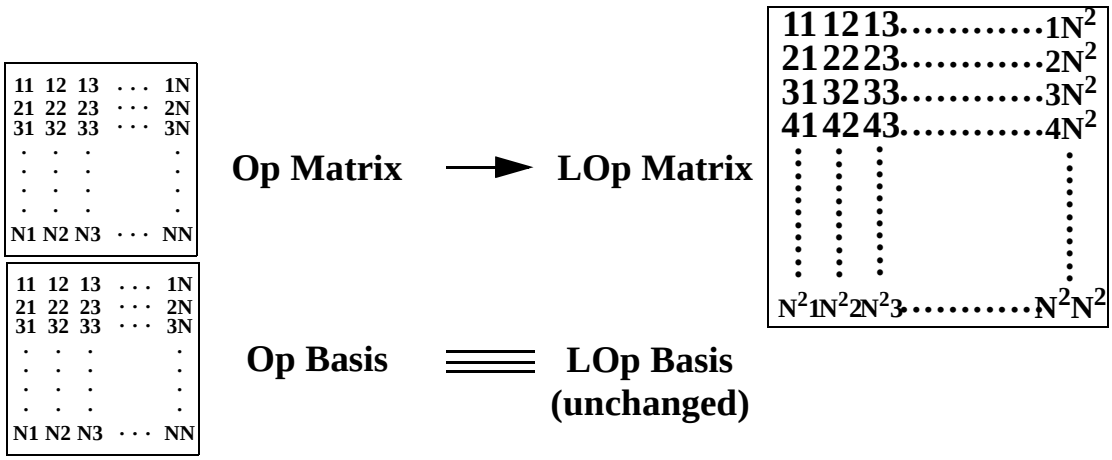

Figure 10-2 - When a superoperator is formed from an operator (or operators) its dimension is the square of the operator. On the other hand, the basis for both remain in the Hilbert space having the operator dimension.

Operator matrices are smaller in size than related superoperator matrices. Yet the basis matrices for Op and LOp can be identical and exist in Hilbert space. Since superoperators are usually formed from operators it is not a problem that the superoperator basis remains in Hilbert space. Operators will always be converted to the superoperator basis with the Hilbert space transformation matrix, *i.e.* the basis. An operator in an arbitrary basis (AB superscript) can be converted into the basis of the superoperator (SB superscript) by first transforming into the default basis (DB superscript) and then into SB.

$$(\boldsymbol{U}^{AB})^{\dagger}\boldsymbol{Op}^{AB}\boldsymbol{U}^{AB} = \boldsymbol{Op}^{DB} \tag{14-1}$$

$$\boldsymbol{U}^{SB}\boldsymbol{Op}^{AB}(\boldsymbol{U}^{SB})^{\dagger} = \boldsymbol{Op}^{SB} \tag{14-2}$$

Here **U** is a transformation matrix or basis matrix (see *Class BASIS*) in Hilbert space, **Op** the Operator matrix in Hilbert space and $\boldsymbol{U}^{SB}$ the basis in which the superoperator resides, also a Hilber space array. Thus, it is generally unnecessary to change the superoperator basis. Unlike general operators, superoperators reside in only one basis at a time.

The Commutation Superoperator

The commutation superoperator is defined to be[1],

$$\Gamma\boldsymbol{A} = [\boldsymbol{Op}, \boldsymbol{A}] = \boldsymbol{OpA} - \boldsymbol{AOp} \tag{14-3}$$

where Γ is a general superoperator (not explicitly a commutation superoperator) and both $\boldsymbol{Op}$ and

1. Ernst, Bodenhausen, and Wokaun, page 23, equation (2.1.81)

$\boldsymbol{A}$ are general operators. The superoperator in this case relates to the operator $\boldsymbol{Op}$ according to

$$\Gamma = (\boldsymbol{Op} \otimes \boldsymbol{E}) - (\boldsymbol{E} \otimes \boldsymbol{Op}^T) \tag{14-4}$$

The superscript T implies the transpose, $\otimes$ the tensor product, and $\boldsymbol{E}$ the identity matrix of dimension equivalent to $\boldsymbol{Op}$ (the Hilbert space dimension.

*The Left and Right Translation Superoperator*s

The commutation superoperator defined in equation (14-3) can also be written as a difference between the left and right translation superoperators[1],

$$\Gamma\boldsymbol{A} = [\boldsymbol{Op}, \boldsymbol{A}] = \boldsymbol{OpA} - \boldsymbol{AOp} = \Gamma^L\boldsymbol{A} - \Gamma^R\boldsymbol{A} \tag{14-5}$$

where Γ^L is the left translation superoperator and Γ^R the right translation superoperator. Again $\boldsymbol{Op}$ and $\boldsymbol{A}$ are general operators. Evidently the two translation superoperators are

$$\Gamma^L = \boldsymbol{Op} \otimes \boldsymbol{E} \qquad \text{and} \qquad \Gamma^R = \boldsymbol{E} \otimes \boldsymbol{Op}^T \tag{14-6}$$

The superscript T implies the transpose, $\otimes$ the tensor product, and $\boldsymbol{E}$ the identity matrix of dimension equivalent to $\boldsymbol{Op}$ (the Hilbert space dimension.

The Double Commutation Superoperator

The double commutation superoperator is defined to be,

$$\Gamma\boldsymbol{A} = [\boldsymbol{Op}1, [\boldsymbol{Op}2, \boldsymbol{A}]] \tag{14-7}$$

where Γ is a superoperator and both $\boldsymbol{Op}1$, $\boldsymbol{Op}2$, and $\boldsymbol{A}$ are general operators. The single commutation superoperator is also defined (see function commutator) and relates to the operator $\boldsymbol{Op}$ according to

$$\Gamma'\boldsymbol{A} = [\boldsymbol{Op}, \boldsymbol{A}] = \boldsymbol{OpA} - \boldsymbol{AOp}\ , \quad \text{where} \quad \Gamma' = (\boldsymbol{Op} \otimes \boldsymbol{E}) - (\boldsymbol{E} \otimes \boldsymbol{Op}^T)\ .$$

The superscript T implies the transpose, $\otimes$ the tensor product, and $\boldsymbol{E}$ the identity matrix of dimension equivalent to $\boldsymbol{Op}$ (the Hilbert space dimension). One can expand the double commutator in terms of single commutators and apply the single commutator superoperators twice.

$$\Gamma\boldsymbol{A} = [\boldsymbol{Op}1, [\boldsymbol{Op}2, \boldsymbol{A}]] = \boldsymbol{Op}1[\boldsymbol{Op}2, \boldsymbol{A}] - [\boldsymbol{Op}2, \boldsymbol{A}]\boldsymbol{Op}1$$

$$\Gamma\boldsymbol{A} = \boldsymbol{Op}1(\Gamma2\boldsymbol{A}) - (\Gamma2\boldsymbol{A})\boldsymbol{Op}1 = [\boldsymbol{Op}1, (\Gamma2\boldsymbol{A})] = \Gamma1\Gamma2\boldsymbol{A}$$

Thus, the double commutator could be produced by the superoperator product in the previous equation, namely

$$\Gamma = \Gamma1\Gamma2\,, \tag{14-8}$$

where Γ is the double commutation superoperator, $\Gamma1$ is the single commutation superoperator for the outside commutator, and $\Gamma2$ is the single commutation superoperator for the inside commutator. Although viable, this is an inefficient way to produce the double commutator superoper-

1. Ernst, Bodenhausen, and Wokaun, page 20, equation (2.1.59)

ator as can be seen from the original single commutator definition.

$$\Gamma 1\Gamma 2 = \Gamma 1\{(\boldsymbol{Op}2 \otimes \boldsymbol{E}) - (\boldsymbol{E} \otimes \boldsymbol{Op}2^T)\}$$

$$\Gamma 1\Gamma 2 = \{(\boldsymbol{Op}1 \otimes \boldsymbol{E}) - (\boldsymbol{E} \otimes \boldsymbol{Op}1^T)\}\{(\boldsymbol{Op}2 \otimes \boldsymbol{E}) - (\boldsymbol{E} \otimes \boldsymbol{Op}2^T)\}$$

$$\Gamma 1\Gamma 2 = (\boldsymbol{Op}1 \otimes \boldsymbol{E})(\boldsymbol{Op}2 \otimes \boldsymbol{E}) - (\boldsymbol{Op}1 \otimes \boldsymbol{E})(\boldsymbol{E} \otimes \boldsymbol{Op}2^T) - (\boldsymbol{E} \otimes \boldsymbol{Op}1^T)(\boldsymbol{Op}2 \otimes \boldsymbol{E}) + (\boldsymbol{E} \otimes \boldsymbol{Op}1^T)(\boldsymbol{E} \otimes \boldsymbol{Op}2^T))$$

Using the distributive property of tensor algebra[1],

$$\Gamma 1\Gamma 2 = (\boldsymbol{Op}1\boldsymbol{Op}2 \otimes \boldsymbol{E}) - (\boldsymbol{Op}1 \otimes \boldsymbol{Op}2^T) - (\boldsymbol{Op}2 \otimes \boldsymbol{Op}1^T) + (\boldsymbol{E} \otimes \boldsymbol{Op}1^T\boldsymbol{Op}2^T) \quad (14\text{-}9)$$

Use of equation (14-9) is computationally more efficient than application of equation (14-8) as demonstrated by the following figure.

Mathematical Operations Necessary for Double Commutator

	Equation (11.2)	Equation (11.4)	Equation (11.5)
Op x Op (Hilbert Space)			2
Op +/- Op (Hilbert Space)	1	2	3
Op x Op (Hilbert Space)	2	4	4
LOp x LOp (Liouville Space)		1	

Figure 10-3 - This table shows the mathematical operations used in the three equations applicable to the computation of the double commutation superoperator.

Equation (14-8) involves 2 single commutator calculations (with (14-4)) as well as one superoperator multiplication step. The Liouville space multiplication is computationally long and not necessary when using equation (14-9). Although (14-9) has two operator multiplications and one more operator subtraction than does (14-8) the total computation needed for these should be much less than the superoperator multiplication[2].

1. The distributive property used here is (A⊗B)(C⊗D)=(AC)⊗(BD).
2. The computational differences will be more pronounced as the Liouville space size increases.

It may be of some use to show the derivation of equation (14-9) pictorially. Considering a single spin 1/2 system, each Hilbert space operator will be a 2x2 array. Equation (14-8) looks schematically as

$$\Gamma 1\Gamma 2 = \left(\boxed{Op1} \otimes \boxed{\begin{matrix}1 & \\ & 1\end{matrix}}\right) \mathbf{x} \left(\boxed{Op2} \otimes \boxed{\begin{matrix}1 & \\ & 1\end{matrix}}\right)$$
$$- \left(\boxed{Op1} \otimes \boxed{\begin{matrix}1 & \\ & 1\end{matrix}}\right) \mathbf{x} \left(\boxed{\begin{matrix}1 & \\ & 1\end{matrix}} \otimes \boxed{Op2^T}\right)$$
$$- \left(\boxed{\begin{matrix}1 & \\ & 1\end{matrix}} \otimes \boxed{Op1^T}\right) \mathbf{x} \left(\boxed{Op2} \otimes \boxed{\begin{matrix}1 & \\ & 1\end{matrix}}\right)$$
$$+ \left(\boxed{\begin{matrix}1 & \\ & 1\end{matrix}} \otimes \boxed{Op1^T}\right) \mathbf{x} \left(\boxed{\begin{matrix}1 & \\ & 1\end{matrix}} \otimes \boxed{Op2^T}\right)$$

When the cross products are performed, the matrix size increases (from Hilbert to Liouville space). The picture then appears as the following.

$$\Gamma 1\Gamma 2 = \boxed{\begin{matrix}Op1_{11}E & Op1_{12}E \\ Op1_{21}E & Op1_{22}E\end{matrix}} \mathbf{x} \boxed{\begin{matrix}Op2_{11}E & Op2_{12}E \\ Op2_{21}E & Op2_{22}E\end{matrix}} - \boxed{\begin{matrix}Op1_{11}E & Op1_{12}E \\ Op1_{21}E & Op1_{22}E\end{matrix}} \mathbf{x} \boxed{\begin{matrix}Op2^T & 0 \\ 0 & Op2^T\end{matrix}}$$
$$- \boxed{\begin{matrix}Op1^T & 0 \\ 0 & Op1^T\end{matrix}} \mathbf{x} \boxed{\begin{matrix}Op2_{11}E & Op2_{12}E \\ Op2_{21}E & Op2_{22}E\end{matrix}} + \boxed{\begin{matrix}Op1^T & 0 \\ 0 & Op1^T\end{matrix}} \mathbf{x} \boxed{\begin{matrix}Op2^T & 0 \\ 0 & Op2^T\end{matrix}}$$

The next step is to perform the sub-matrix multiplications.

$$\Gamma 1\Gamma 2 = \begin{array}{|c|c|}\hline (Op1_{11}\,Op2_{11} + Op1_{12}\,Op2_{21})\,E & (Op1_{11}\,Op2_{12} + Op1_{12}\,Op2_{22})\,E \\ \hline (Op1_{21}\,Op2_{11} + Op1_{22}\,Op2_{21})\,E & (Op1_{21}\,Op2_{12} + Op1_{22}\,Op2_{22})\,E \\ \hline \end{array} - \begin{array}{|c|c|}\hline Op1_{11} \cdot Op2^T & Op1_{12} \cdot Op2^T \\ \hline Op1_{21} \cdot Op2^T & Op1_{22} \cdot Op2^T \\ \hline \end{array}$$

$$- \begin{array}{|c|c|}\hline Op2_{11} \cdot Op1^T & Op2_{12} \cdot Op1^T \\ \hline Op2_{21} \cdot Op1^T & Op2_{22} \cdot Op1^T \\ \hline \end{array} + \begin{array}{|c|c|}\hline Op1^T\,Op2^T & 0 \\ \hline 0 & Op1^T\,Op2^T \\ \hline \end{array}$$

It is thus obvious how the distributive property applies.

$$\Gamma 1\Gamma 2 = \left(\boxed{Op1} \times \boxed{Op2} \otimes \boxed{\begin{matrix} 1 & \\ & 1 \end{matrix}} \right) - \boxed{Op1} \otimes \boxed{Op2^T}$$

$$- \boxed{Op2} \otimes \boxed{Op1^T} + \boxed{\begin{matrix} 1 & \\ & 1 \end{matrix}} \otimes \left(\boxed{Op1^T} \times \boxed{Op2^T} \right)$$

This is of course equivalent to our working equation (14-9).

$$\Gamma 1\Gamma 2 = (\boldsymbol{Op}1\boldsymbol{Op}2 \otimes \boldsymbol{E}) - (\boldsymbol{Op}1 \otimes \boldsymbol{Op}2^T) - (\boldsymbol{Op}2 \otimes \boldsymbol{Op}1^T) + (\boldsymbol{E} \otimes \boldsymbol{Op}1^T\boldsymbol{Op}2^T)$$

Of final note is the situation which occurs when the two operators are equivalent, i.e. when

$$\boldsymbol{Op} = \boldsymbol{Op}1 = \boldsymbol{Op}2.$$

The equation then becomes

$$\Gamma = \Gamma 1\Gamma 1 = (\boldsymbol{Op}^2 \otimes \boldsymbol{E}) - 2(\boldsymbol{Op} \otimes \boldsymbol{Op}^T) + (\boldsymbol{E} \otimes \boldsymbol{Op}^{T2}) \qquad (14\text{-}10)$$

which requires one less cross product to be taken than in equation (14-9).

The Unitary Transformation Superoperator

The unitary transformation superoperator is defined to be[1],

$$\Gamma \boldsymbol{A} = \boldsymbol{Op}\boldsymbol{A}\boldsymbol{Op}^{-1} \qquad (14\text{-}11)$$

1. Ernst, Bodenhausen, and Wokaun, page 24, equation (2.1.83)

where Γ is a general superoperator (not explicitly a unitary transformation superoperator) and both $\boldsymbol{Op}$ and $\boldsymbol{A}$ are general operators. The superoperator in this case relates to the operator $\boldsymbol{Op}$ according to

$$\Gamma = \boldsymbol{Op} \otimes \boldsymbol{Op}^* \tag{14-12}$$

The superscript * implies the complex conjugate and $\otimes$ the tensor product.

Superoperator Exponentials

Whenever superoperators mix with operators it is perferable to perform any basis transformations in the Hilbert space, *i.e.* on the operators not the superoperators. This is simply because of the vast increase in size of the matrices involved. This desire motivated class superoperator maintaining its basis in Hilbert space and not Liouville space.

Seemingly contrary to this, there are times when one desires the superoperator in a different basis and this is in fact allowed. This situation typically arises in computation of the following exponential.

$$exp[(\boldsymbol{LOp})t] \tag{14-13}$$

To compute this exponential the superoperator must be placed in its eigenbasis, where the superoperator matrix is diagonal. In this instance, there will be a basis matrix in Liouville space which will convert the eigenbasis (diagonal form) back to the initial basis.

$$(\boldsymbol{U}_L^{\boldsymbol{SB}})^\dagger \boldsymbol{LOp}^{\boldsymbol{EB}} \boldsymbol{U}_L^{\boldsymbol{SB}} = \boldsymbol{LOp}^{\boldsymbol{SB}} \tag{14-14}$$

Here $\boldsymbol{U}_{\boldsymbol{L}}$ is a Liouville space transformation matrix or basis matrix. Superscript SB is used for the original superoperator basis, and superscript EB for the eigenbasis. $\boldsymbol{LOp}^{\boldsymbol{EB}}$ is the superoperator in its eigenbasis and is diagonal. The exponential now becomes

$$exp[(\boldsymbol{LOp}^{\boldsymbol{SB}})t] = (\boldsymbol{U}_L^{\boldsymbol{SB}})^\dagger \{exp[(\boldsymbol{LOp}^{\boldsymbol{EB}})t]\} \boldsymbol{U}_L^{\boldsymbol{SB}} \quad . \tag{14-15}$$

When this exponential is to be determined for different times it is essential that the superoperator be stored in its eigenbasis and this mandates the storage of the corresponding basis in Liouville space.